Geometric Approach and Taboo Search for Scheduling Flexible Manufacturing Systems

Yazid Mati, Nidhal Rezg, Xiaolan Xie

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Geometric Approach and Taboo Search for Scheduling Flexible Manufacturing Systems

Yazid Mati - Nidhal Rezg - Xiaolan Xie

N° 4137
Mars 2001

THÈME 4

Geometric Approach and Taboo Search for Scheduling Flexible Manufacturing Systems

Yazid Mati, Nidhal Rezg and Xiaolan Xie

INRIA / MACSI Project & LGIPM
ENIM - ILE DU SAULCY, 57045 Metz, France.
Tel: (33)-3-87 34 67 34. Fax: (33)-3-87 34 69 35. E-mail: {ymati, nrezg, xie}@loria.fr

Abstract

This paper addresses the scheduling and deadlock avoidance of a class of automated manufacturing systems. In such systems, a set of jobs is to be performed on a set of resources and each job requires several operations. An operation may require several types of resources with several units of each type. Further, upon the completion of an operation, its related resources cannot be released until resources needed for the next operation become available. One important characteristic of such systems is the possibility of deadlock. The scheduling problem deals with the allocation of resources such that jobs are completed within a minimal makespan and deadlocks are avoided. We extend the classical geometric approach to solve the two-job case of our model. A greedy algorithm based on this result and the taboo search heuristic is then developed for the general case. Numerical results show that the proposed algorithm is fast and provides good schedules.

Keywords: Manufacturing systems, Scheduling, Deadlock avoidance, Geometric approach, Taboo search

une approche de recherche tabou pour l'ordonnancement sans blocage des systèmes manufacturiers automatisés

Résumé:

Cet article traite le problème d'ordonnancement d'une classe des systèmes manufacturiers automatisés. Un nouveau modèle des systèmes manufacturiers est proposé. Dans ce modèle, un ensemble de travaux doit être exécuté et chaque travail demande une séquence d'opérations. Chaque opération peut nécessiter plusieurs ressources. Après la fin d'une opération, les ressources nécessaires pour l'opération suivante du même travail ne sont pas libérées et les ressources restantes ne peuvent pas être libérées jusqu'au début de l'opération suivante. Le problème d'ordonnancement consiste à trouver un ordre de passage sur les ressources pour chaque opération afin d'éviter les blocages et de minimiser le makespan. La représentation classique par le graphe disjonctif est généralisée pour modéliser le problème d'ordonnancement. Un algorithme de recherche tabou est ensuite proposé en utilisant une nouvelle structure du voisinage définie par deux mouvements de base: la permutation des arcs disjonctifs des chemins critiques et un mouvement de recouvrement du blocage si le premier échoue. Les résultats numériques présentés montrent l'efficacité de l'algorithme proposé.

Mots clés: Ordonnancement, Systèmes Manufacturiers, Blocage, Recherche Tabou

1. Introduction

Flexible Manufacturing Systems (FMSs) have been introduced to improve the productivity of classical job shops while retaining their flexibility. The key reason of their success is the fact that these systems are capable of producing a broad variety of products and changing their characteristics quickly and frequently. This is realized by efficient use of machines, tools, robots and modern transportation resources such as Automated Guided Vehicles (AGVs) and Conveyors. Automated storage devices with a very limited (maybe zero) buffer space are also frequently used. High costs of these manufacturing resources result in their limited availability. One of the most important and difficult issues arisen when managing these systems is the allocation of manufacturing resources such that systems can operate with minimal cost.

In this paper a scheduling model that incorporates the above characteristics is proposed. In this model a set of jobs have to be performed. Each job requires a sequence of operations to be accomplished. An operation may need more than one resource with multiple units of each resource. Most importantly, upon the completion of an operation its related resources cannot be released until the resources needed for the next operation become available. The objective is to find a deadlock-free schedule that minimizes the completion time of all operations. This model generalizes the multiprocessor job shop scheduling problem, which received considerable attention in the literature (Dauzère-Pérès and Paulli, 1997). Extensions, such as the routing flexibility and assembly \ disassembly operations can easily be integrated in our model.

Being an extension of the classical job shop problem, which is known to be NP-hard, the scheduling problem considered in this paper is NP-hard as well. Comparing with the huge scheduling literature (Blazewicz, *et al.*, 1996; GOTHA, 1993), scheduling problem under real-life constraints arisen in automated manufacturing systems has received little attention. Existing methods take into account very simple behaviors representing simple machine, flow shop and job shops, etc. Exceptions include work on management of transportation resources, tools and robots (Blazewicz and Finke, 1994; Hall and Sriskandarajah, 1996; Sethi et al., 1992).

As noticed by Ramaswamy and Joshi (1996), the planning and control of strongly automated systems differs considerably from that in traditional scheduling literature. In fact, complex product structure and complex resource interactions make the system deadlock-prone, and a poor management may lead to undesirable deadlocks. Traditional approach to this problem is to solve first the scheduling problem without deadlock consideration and then try to realize the schedule using some real-time control policies. Various solutions were proposed to handle deadlock in automated manufacturing systems (Banaszak and Krogh, 1990; Chu and Xie, 1997; Ezpeleta, et al., 1995; Fanti, et al., 1997). They are deadlock prevention, detection and avoidance methods. The deadlock detection is based on static policy of resource allocation in order to eliminate deadlock situations for each state of the system. On the other hand, the deadlock avoidance is a dynamic strategy that determines dynamically admissible allocations. Deadlock detection/avoidance is an NP-hard problem, due to the lack of fast and efficient methods to determine all deadlock situations.

As a consequence of the above discussions, it is our belief that the separate consideration of scheduling and deadlock avoidance may lead to unsatisfactory performance. In this paper, we propose a heuristic algorithm that takes into account deadlock situations during the

construction of the schedules. The manufacturing systems under consideration is presented and justified. The well-known geometric approach for the two-job shop problems is extended to solve instance of the two-job shop with blocking. We then introduce the notion of the combined job that group two scheduled jobs into a single one. Based on these results, a greedy heuristic is proposed to compute the schedule for any number of jobs according to a given job sequence. Taboo search is then used to improve the greedy heuristic.

Up to now, most researchers in the area of flexible manufacturing systems focus on exact methods. The first class of these methods is characterized by the work of Ramaswamy and Joshi (1996), who provided a mixed integer linear programming model for obtaining deadlock-free schedules for an automated workstation with machines, transportation resources and limited buffers. The second class adopts the Petri nets to model the scheduling problem and uses the so-called A^* algorithm and the reachability graphs to obtain optimal solutions (Jeng, et *al.*, 1996; Lee and DiCesare, 1994). Despite the fact that these methods lead to optimal solutions, they are very memory and time consuming and are not applicable in practical situations. Heuristic methods were proposed in (Proth and Xie, 1996) for scheduling manufacturing systems with complex product structures and simple resource behaviors, in which each operation requires one resource and each operation is followed by an unlimited buffer. Scheduling problems of cyclic manufacturing systems were also formulated and solved using heuristic methods in (Camus, 1997; Hillion, et *al.*, 1987). Most related to our work are the works of Damasceno and Xie (1998, 1999). They developed a heuristic approach based on dynamic programming and used the Petri net as a tool for deadlock detection.

The rest of the paper is organized as follows. Section 2 describes the manufacturing system under consideration. In Section 3, the classical geometric approach is reviewed. Section 4 proposes an extension of this approach for the two-job shop with blocking. Section 5 presents the greedy method for the general case. In Section 6 the taboo search heuristic is developed to improve the greedy method. Before giving conclusion in Section 8, Section 7 presents numerical results on some instances from the literature.

2. Problem setting

In this section, the manufacturing system under consideration is described. This system is called Multi-resource Job Shop with Blocking or MJSB for short. One motivation of this system is to provide a general framework for modeling automated manufacturing systems.

Formally speaking, the multi-resource job shop with blocking is composed of a set of resources $\Re=\{1, 2,..., r, ..., m\}$, where m is the number of types of resources. Each resource r is available with a given quantity Q_r. A set of N jobs $\Im=\{J_1, J_2,..., J_i,..., J_N\}$ must be performed. Each job J_i consists of a sequence of operations $\{O_{i1}, O_{i2},..., O_{in_i}\}$, where n_i is the number of operations of the job. According to the terminology used in the scheduling theory, the sequence of operations is also called the manufacturing process of job J_i. Operations of job J_i must be performed according to the manufacturing process of that job. An operation O_{ik} requires p_{ik} time units, and must be executed without interruption. Further, it needs several types of resources $R_{ik} \subset \Re$, with several units B_{ik}^r of each resource $r \in R_{ik}$. In this paper the following form is adopted to represent the manufacturing process of a job J_i= $\{(R_{i1}, B_{i1}, p_{i1}), (R_{i2}, B_{i2}, p_{i2}),..., (R_{in_i}, B_{in_i}, p_{in_i})\}$. For example, J_i = {($2M_1$, M_2, 2), ($2M_2$, 3)}

is a job of two operations. The first operation requires 2 units of resource M_1 and one M_2 and 2 time units. The second operation requires 2 units of resource M_2 and 3 time units.

Most importantly, we make the following assumption regarding the release of resources after the completion of an operation:

> *Hold while wait assumption:* After the completion of an operation O_{ik} of a job J_i, all resources in R_{ik} are held by job J_i until resources needed for the subsequent operation $O_{i,k+1}$ become available. At this point, $\text{Max}\left\{0, B^r_{ik} - B^r_{i,k+1}\right\}$ units of each resource r that are not needed in operation $O_{i,k+1}$ are released and of course, the other resources are still held by job J_i for operation $O_{i,k+1}$.

Although our definition of jobs is not different from that of the classical job shop models, our definition of resources is wider than the classical ones. Of course, resources considered in our model include machines, tools, operators, buffer spaces and transportation resources. It also allow modeling technical constraints such as (i) mutual exclusive presence in the system of jobs, (ii) maximal number of jobs in a critical area and (iii) the number of jobs in a kanban-controlled system.

Another departure from the classical job shop models is the definition of operations. More precisely, operations considered in our model include not only classical machining operations but also material handling operations and wait. For instance, an operation O_{ik} such that $R_{ik} = \varnothing$ and $p_{ik}=0$ corresponds to a job waiting for its next operation. Such an operation is termed a waiting operation. Our definition of operations can be considered as a detailed description of different phases of a job.

The introduction of "Hold while wait" constraint, and the above definitions of resources and operations, provides a general framework for modeling automated manufacturing systems with various resources and real-life technical constraints. We illustrate this claim by showing that our model includes but not limited to the following systems:

- Classical job-shops. The MJSB representation of each job is $\{(M_{i1}, p_{i1}), (\varnothing,0), (M_{i2}, p_{i2}), (\varnothing,0), \ldots\}$ where $\{M_{i1}, M_{i2}, \ldots\}$ is the sequence of machines to be visited by the job. Note that the MJSB representation explicitly describes the assumption of infinite buffer space by introducing a waiting operation $(\varnothing,0)$.
- Production lines without buffer space (Hall, *et al.*, 1996). The MJSB representation of each job is $\{(M_1, p_{i1}), (M_2, p_{i2}),\ldots, (M_N, p_{iN})\}$ where $\{M_1, M_2,\ldots, M_N\}$ is the series of machines.
- Robotic cells composed of a set of resources arranged around a robot for loading/unloading (Sethi, *et al.*, 1992). The MJSB representation of a job is $\{(R, \Delta), (M_1, p_{i1}), (R, \Delta), (M_2, p_{i2}),\ldots\}$ where *R* denotes the robot, $M_1, M_2,\ldots, M_m$ the machines to visit, and Δ the load/unload time.
- Conveyor for transportation of parts between two machines M_1 and M_2. The MJSB representation of a job is $\{\ldots, (M_1, p_{i1}), (\varnothing, \Delta), (M_2, p_{i2}),\ldots\}$ where Δ is the transportation delay.
- Flexible manufacturing systems (FMS) that are job shops but use AGVs for transporting jobs inside the system. The MJSB representation of a job is $\{(AGV, M_{i1}, p_{i1}), (AGV, M_{i2}, p_{i2}),\ldots\}$.

These examples provide justification for our "Hold while wait" constraint. To see this, let us notice that (i) a machine completing a part in a production line without buffers remains occupied until the part moves into the downstream machine, (ii) a machine completing a part in a robotic cell becomes free only after the unloading of the part and (iii) an AGV is needed to start the processing of a part in an FMS and it is released only after the completion of all operations of the part.

An important issue encountered with MJSB is the coordination of the set of resources. Since the existence of "Hold while wait" constraint, deadlock situations may occur in the system, and a poor coordination may lead the system to undesirable states. For example, in a robotic cell, a deadlock arises if the robot carries a part to a machine, which is working on an another part. Under such a situation, the robot cannot unload the part since the machine is not free while the machine will need the robot to download the part it is working on.

The scheduling problem consists in choosing an input sequence σ_r of operations into each resource $r \in \Re$ and starting times S_{ik} of all operations such that the total completion time called makespan C_{Max} is minimized and deadlocks are avoided. The scheduling problem can be formulated as follows:

$$
(P)\quad \begin{cases}
\text{Min} \quad C_{Max} & \\
\text{subject to :} & \\
\quad S_{i,k+1} - S_{ik} \geq p_{ik} & \forall\, 1 \leq i \leq N, \forall\, 1 \leq k \leq n_i - 1 \qquad (1)\\
\quad C_{Max} \geq S_{in_i} + p_{in_i} & \forall\, 1 \leq i \leq N \qquad (2)\\
\quad \sum_{(i,k)} B_{ik}^r \, \mathbf{1}_{\{S_{ik} \leq t < S_{i,k+1}\}} \leq Q_r & \forall \text{ time period } t,\ \forall r \qquad (3)\\
\quad \text{The input sequences } \sigma_r \text{ do not lead to deadlock} & \qquad (4)\\
\quad S_{ik} \geq 0 & \forall\, 1 \leq i \leq N, \forall\, 1 \leq k \leq n_i
\end{cases}
$$

Constraints (1) ensure the precedence relations of jobs. Constraints (2) define the jobs completion times. Constraints (3) avoid resource capacity violation. In general case when the resources are available in multiple units, the deadlock-freeness cannot be easily formulated in terms of the decision variables. Note that it is possible to formulate the deadlock-freeness in special cases with single units (see Ramaswamy and Joshi, 1996).

3. The two-job shop problem of classical job-shop

The job shop scheduling problem is one of the most difficult problems encountered in combinatorial optimization. Although this problem is NP-hard in general case, there are special cases that are efficiently solvable. The two-job shop problem is one of these few cases.

In this section, we review the geometric approach to solve the two-job shop problem when the objective functions $f(C_1, C_2)$ to be minimized are regular, i.e. monotone non decreasing functions of the completion times C_1 and C_2 of both jobs. We consider here the makespan

minimization to explain this approach. This problem can be formulated as a shortest path in a network (Brucker, 1988), and consequently can be solved in polynomial time.

The two-job shop scheduling problem can be stated as follows: Assume that we have two jobs $J_i\{i=1,2\}$ to be scheduled, with manufacturing processes $\{O_{i1}, O_{i2}, ..., O_{in_i}\}$, where n_i is the number of operations of job J_i. Each operation O_{ij} has to be executed on a given machine without interruption, and needs p_{ij} time units. A machine can perform at most one operation at a time. The scheduling problem consists in sequencing the operations on the machines such that the makespan is minimized.

This problem can be represented in a 2-dimension plane with obstacles, which represent the resources conflict between operations of the two jobs, and the makespan minimization problem becomes a shortest path problem in this 2-demension plane. More precisely,

- Each job J_i is represented by an axe with n_i intervals according to the manufacturing process. Each interval corresponds to an operation O_{ij} and has a length of p_{ij},
- Intervals O_{1j} and O_{2k} form an obstacle if O_{1j} and O_{2k} share the same resource (see Figure 1),
- The horizontal and the vertical crossing the final point $F=\left(\sum_{k=1}^{n_1} p_{1k}, \sum_{k=1}^{n_2} p_{2k}\right)$ are considered as the final obstacle.

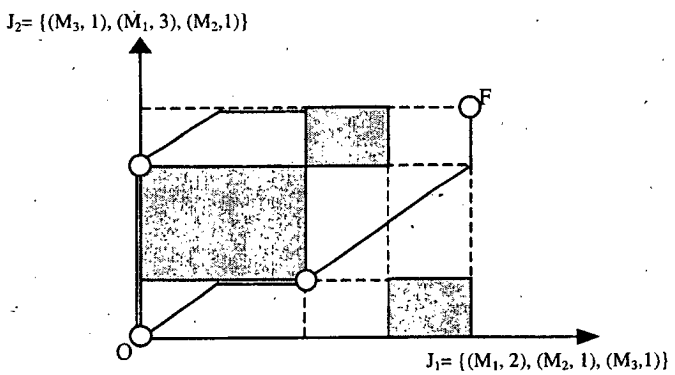


Figure 1. A shortest path in the plane

A feasible solution of the two-job shop problem corresponds to a path from the origin point O to the final point F. The path is composed of horizontal segments (only an operation of J_1 is processed), vertical segments (only an operation of J_2 is processed) and diagonal segments (both jobs are processed). The path avoids the interior of any obstacle as the corresponding operations require the same machine, they cannot be processed simultaneously and cannot be interrupted. The length of a horizontal or vertical segment is equal to its usual length while the length of a diagonal segment is equal to the length of its projection in any axe, which is the time spent for the simultaneous processing of two operations.

The shortest path problem in the plane can be transformed into an unrestricted shortest path problem in an acyclic network N=(V, E, d), where the set of nodes V corresponds to the origin O, the final point F and the North-West and South-East corners of the obstacles. Each node v_i has at most two successors obtained by going diagonally until hitting an obstacle D. If the obstacle D is the final obstacle, the node F is the only successor of node v_i, otherwise the

NW and SE corners, C_y and C_x of obstacle D, are immediate successors of v_i (see Figure 2). The distance between v_i and C_x (resp. C_y) is the projection along axe x (resp. y).

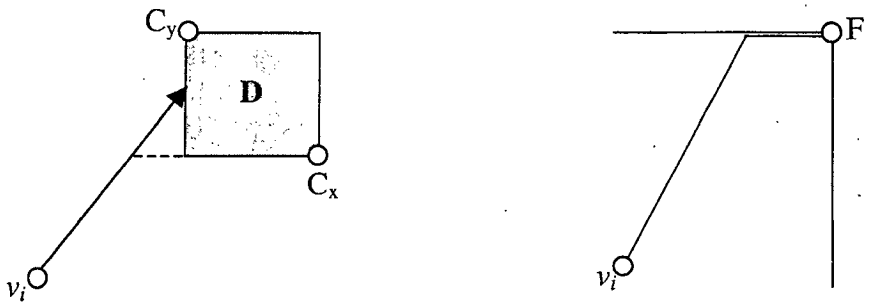


Figure 2. Successors of a node v_i

It has been shown in (Brucker, 1988) that the network N can be constructed in O(r log r) steps where r is the number of the obstacles, and may be as large as $O(n_1 n_2)$. On the other hand, the shortest path in the constructed acyclic network can be obtained in O(r) steps. Consequently, the two-job shop problem can be solved in O(r log r) time. We notice that the geometric approach also works for other instances of the two-job shop including instances where the repetition of machines is allowed, the preemption, and the multi-purpose machines that have identical (uniform) speeds. For more details on the extensions of the geometric approach, the reader is referred to (Brucker, 1998).

4. Two-job MJSB

As mentioned in Section 2, the most salient feature of the multi-resource job shop with blocking is the "Hold while wait" constraint. Due to this constraint, deadlock situations may occur in the system and the management of the resources becomes more difficult.

The goal of this section is to propose an extension of the geometric approach described in the previous section for scheduling and deadlock avoidance of the two-job case of multi-resource job shop with blocking (2-job MJSB). The extended approach has the advantage of taking into account deadlock situations during the construction of the schedule, and it differs from the classical geometric approach in two major aspects. The first difference is related to the definition of the obstacles, while the second is the construction of the network N=(V, E, d). In the following, we discuss in details how to define the obstacles and construct the network N=(V, E, d) for the 2-job MJSB.

4.1 Nature of obstacles

In the classical job shop problem, the obstacles correspond to resource conflicts between operations of the two jobs, and a feasible path consists of a sequence of horizontal, vertical and diagonal segments that avoid the interior of the obstacles.

In order to deal with the deadlock avoidance problem of 2-job MJSB, additional obstacles must be added. These obstacles restrict the paths going from the origin O to the final point F to avoid resource capacity violation and deadlock states.

In order to take into account both resource conflicts and deadlock situations in the determination of the obstacles, a backward dynamic programming algorithm is proposed. This

algorithm uses Boolean variables $I(k_1, k_2)$, $k_1=1, 2,\ldots, n_1+1$, $k_2=1, 2,\ldots, n_2+1$ defined as follows:

$$I(k_1,k_2)=\begin{cases}1, & \text{if operations } O_{1k_1} \text{ and } O_{2k_2} \text{ have resource conflict,}\\ & \text{or a deadlock situation will be reached starting}\\ & \text{from state } (k_1,k_2),\\ 0, & \text{otherwise.}\end{cases}$$

Using these variables, all the obstacles can be determined using a recursive equation that can be written in the following form:

$$I(k_1,k_2)=\begin{cases}1, & \text{if } B^r_{1k_1}+B^r_{2k_2}>Q_r \text{ for some } r,\\ I(k_1,k_2+1)\wedge I(k_1+1,k_2), & \text{otherwise.}\end{cases}$$

where Q_r is the number of resources of type r, B^r_{ik} is the number of resources r needed for operation O_{ik}. By convention, $I(k_1, n_2+1)=0$, for all $k_1=1, 2,\ldots, n_1+1$ and $I(n_1+1, k_2)=0$, for all $k_2=1, 2,\ldots, n_2+1$.

Consider an example of two jobs defined as follows:

$$J_1=\{(R_1, 1), (R_1, R_2, 1), (R_3, 1)\},$$
$$J_2=\{(R_3, R_4, 1), (R_1, 1), (R_3, 1)\},$$

where there is only one copy of each type of resources. The above dynamic programming approach yields obstacles that are shown in Figure 3. The hashed obstacles correspond to the resource conflicts, while the dark obstacles correspond to deadlock states. For example, the rectangle defined by operations O_{11} and O_{21} is allowed from the resource availability point of view, but if the system enters into this state, deadlock will occur whatever the schedule of the remaining operations, and hence it must be forbidden.

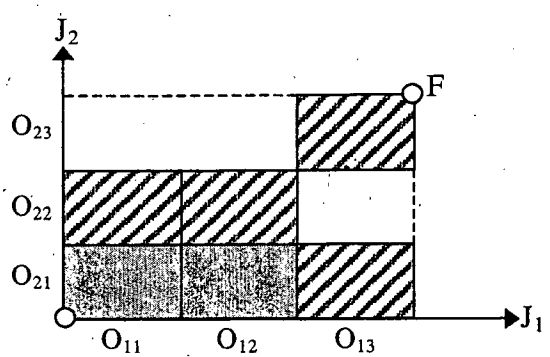


Figure 3. The obstacles constructed by the dynamic programming

Finally, the above algorithm adds only obstacles that lead to deadlock situations and allows determining the obstacles in $O(n_1 \times n_2)$ time. More precisely:

Property 1: The dynamic programming approach is correct and the obstacles can be determined in $O(n_1 \times n_2)$ steps.

Proof: we prove that the dynamic programming approach adds only additional obstacles that lead to deadlock situations. The proof is done by reduction on $k_1+k_2 = n_1+1 + n_2+1$, $n_1+1 + n_2$, ..., 1, 0. Clearly the property holds for $k_1+k_2 = n_1+1 + n_2+1$. Assume that it holds for all $k_1+k_2 \geq k$, i.e. (k_1, k_2) is a deadlock state or a state with resource conflict iff $I(k_1, k_2) = 1$. Consider any state (k_1, k_2) such that $(k_1, k_2) = k-1$. If the resource capacity constraint does not hold in state (k_1, k_2), the property clearly holds since $I(k_1, k_2) = 1$. Otherwise, $I(k_1, k_2) = I(k_1+1, k_2) \wedge I(k_1, k_2+1)$. Since only two jobs are concerned, J_1 proceeds to its next operation first or J_2 proceeds to its next operation first. The next state is either (k_1+1, k_2) or (k_1, k_2+1). State (k_1, k_2) is a deadlock state iff both (k_1+1, k_2) and (k_1, k_2+1) is either a blocking state or a state with resource conflict. By reduction assumption, state (k_1, k_2) is a deadlock state iff $I(k_1+1, k_2) = 1$ and $I(k_1, k_2+1) = 1$, i.e. iff $I(k_1, k_2) = 1$. As a result, the property holds for any state (k_1, k_2) such that $(k_1, k_2) = k-1$ and the property is proved. Q.E.D.

4.2 Construction of the network for computing optimal schedule

As for classical job-shop, a feasible solution of the two-job MJSB problem corresponds to a path from the origin point O to the final point F. The path is composed of horizontal segments (only an operation of J_1 is processed), vertical segments (only an operation of J_2 is processed) and diagonal segments (both jobs are processed). The length of a horizontal or vertical segment is equal to its usual length while the length of a diagonal segment is equal to the length of its projection in any axe. The path avoids the interior of any obstacle.

In contrast to the classical job-shop case, the upper boundary and the right boundary of any obstacle (k_1, k_2) become also forbidden as a result of the "Hold while wait" constraint. This is because at both boundaries, all resources related to the operation O_{2,k_2} (resp. O_{1,k_1}) are still held since job J_2 (resp. J_1) does not yet start operation O_{2,k_2+1} (resp. O_{1,k_1+1}) at the upper (resp. right) boundary.

The main step in the determination of the optimal schedule using the geometric approach is the construction of the network N=(V, E, d), where the shortest path must be determined. This construction transforms the restricted shortest path problem in the plane into the one of an unrestricted shortest path in the network.

Obviously, considering NW and SE corners of all rectangles of the plane as nodes of N=(V, E, d) guaranties the optimal path from O to F to be found. Unfortunately, this increases the number of nodes (arcs) of the network N=(V, E, d) and makes the determination of the shortest path very slow. Consequently, the problem of determining the corners that are sufficient to give the optimal path becomes crucial.

As for classical job-shop case, the nodes of the network correspond to the NW and SE corners of all obstacles. Concerning the definition of successors, it has been shown that for classical job-shops, only NW and SE corners of the obstacles obtained when going diagonally are sufficient to guarantee the optimality. This is not true when the "Hold while wait" constraint is introduced. More precisely, in the MJSB case, the number of successors of a node v_i can be greater than 2. In our MJSB model, all NW (resp. SE) corners of obstacles, that can be reached from a node v_i by first going diagonally and then vertically (resp. horizontally), should be considered as successors of v_i. The definition of the length $d(v, v')$ of any arc (v, v') is similar to that of classical job-shop, i.e. it is equal to the projection along axe X (resp. Y) if

v' is obtained from v by first going diagonally and then horizontally (resp. vertically). In Figure 4, SE_1, NW_2, NW_3 and SE_3 are successors of v_i.

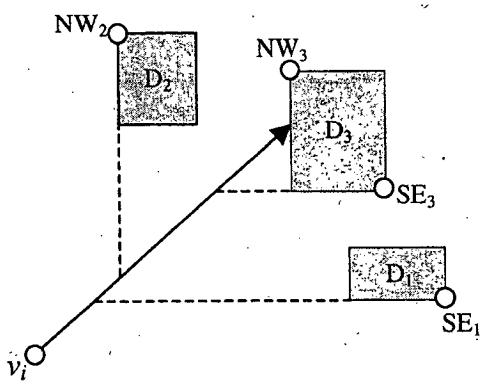


Figure 4. Successors of a node v_i

Now, we show that the nodes of the network obtained by the above construction are sufficient to determine the optimal solution for the two-job MJSB problem. This is giving by the following theorem.

Theorem 1: A shortest path from O to F in the constructed network N=(V, E, d) corresponds to a shortest path on the plane with obstacles defined in Section 4.1. Hence it corresponds to an optimal schedule.

Proof: From the definition of the network N, any path from O to F in N corresponds to a path P from O to F on the plane which avoids the interior, upper boundary and right boundary of all obstacles. P consists of arcs, as shown in Figure 5.

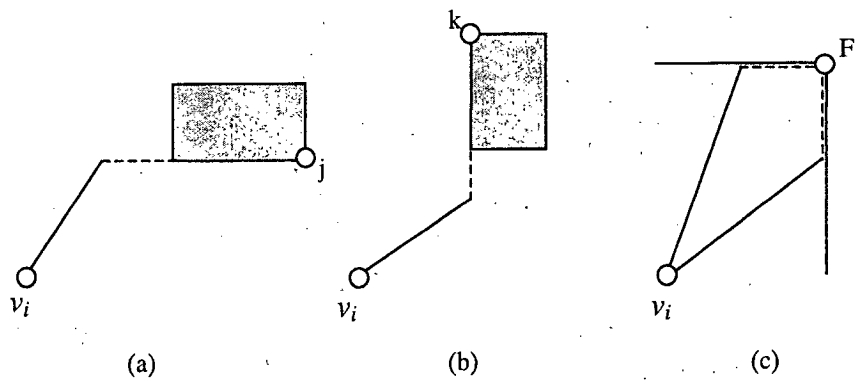


Figure 5. Arcs of the network

In the following, we prove that there exists a shortest path in the plane with obstacles that consists of a sequence of arcs of Figure 5 that are called *network arcs*. This clearly concludes the proof.

Among all shortest paths on the plane with obstacles, consider the one P^* that has the maximal number of consecutive network arcs starting from O. If P^* consists of only network arcs, then the proof is completed. Otherwise, let v_i be the terminating node of these network arcs. From the definition of P^*, the part of the path P^* that follows node v_i is not a network arc. Before going further, let us examine the different ways a feasible path P avoids an obstacle.

From Figure 6, it is clear that there are only two ways to avoid an obstacle D, since a path consists of only horizontal, vertical and diagonal segments. The first way is to cross the horizontal l of the lower boundary of D by some points on the right segment of SE corner j including j. The second way is to cross the vertical l' of the left boundary of D by some points on the upper segment of point k including k. The first way will be called SE avoidance and the second way the NW avoidance.

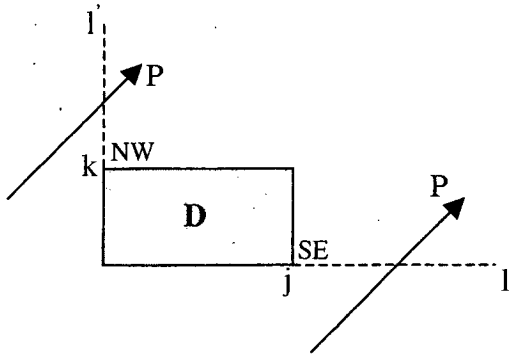


Figure 6. Two ways of avoiding an obstacle

Let us come back to the path P^*. Let D_1 be the obstacle hit when going diagonally starting from node v_i. Assume that D_1 is avoided by path P^* using SE avoidance. The proof for the NW avoidance is similar and omitted.

Let l_1 be the horizontal of the lower boundary of D_1. It is crossed by the diagonal (v_i, s) at s_1 and by path P^* at point t_1 (see Figure 7).

Case 1: the path (s_1, t_1) is a feasible segment avoiding all obstacles. In this case, we can replace the part of P^* between v_i and t_1 by segments (v_i, s_1, t_1) without increasing the length of P^*. However, the new path has one extra network arc (v_i, s_1, SE_1) following v_i. This contradicts the maximality assumption of P^*.

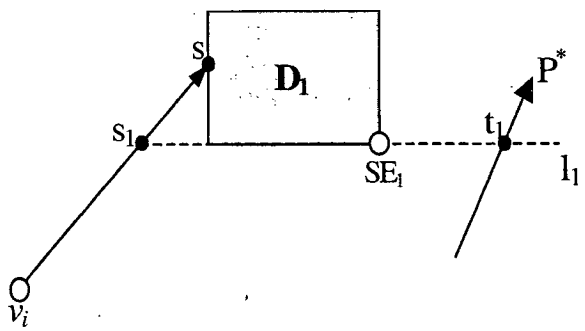


Figure 7. Case 1

Case 2: the path (s_1, t_1) is not a feasible segment. Then there exist obstacles when going from s_1 to t_1 (see Figure 8). Let D_2 the first obstacle met when going from s_1 to t_1.

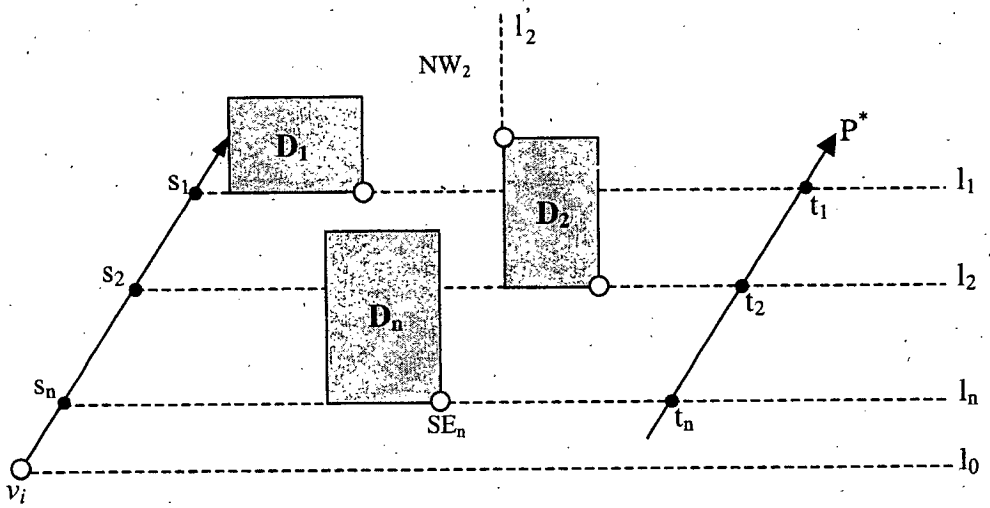


Figure 8. Case 2

Clearly, path P^* avoids D_2 by SE avoidance as well. Otherwise, P^* crosses the vertical l_2' by some points in the upper segment of NW_2 which is beyond the horizontal l_1 and hence P^* cannot cross l_1 at t_1. This is a contradiction and proves that P^* avoids D_2 by SE avoidance. Let l_2 be the horizontal of the lower boundary of D_2. It is crossed by the diagonal at s_2 and P^* at t_2. Clearly, l_2 is lower than l_1. Further, l_2 is not lower than the horizontal l_0 crossing v_i since, otherwise, P^* goes down to l_2 starting from v_i which is a contradiction.

If (s_2, t_2) is not a feasible segment, then there exists an obstacle D_3 on (s_2, t_2). Repeating the above process leads to an obstacle D_n with the following properties: (i) the horizontal l_n of the lower boundary of D_n is between l_1 and l_0; (ii) D_n is avoided by P^* by SE avoidance; (iii) the segment (s_n, t_n) is feasible. Replacing the part of P^* between v_i and t_n by segments (v_i, s_n, t_n) leads to a path with one extra network arc (v_i, s_n, t_n). This contradicts the maximality assumption of P^*.

To summarize, the maximality assumption of P^* is contradicted in all cases. As a results, by contradiction, a point v_i on P^* that is not followed by a network arc does not exist and P^* consists of only network arcs. Q.E.D.

4.3 Complexity of the 2-job MJSB scheduling problem

In this section, we give the complexity in the worst case of the geometric approach for solving an instance of the multi-resource two-job shop problem with blocking. This complexity depends on the dynamic programming algorithm developed in Section 4.1, the construction of the network N=(V, E, d) and the algorithm used to find the shortest path in this network.

The number of successors of a node v_i depends on the processing time of operations. An upper bound of the number of successors of v_i is $(n_1 + n_2)$. Further, there are at most $n_1 \times n_2$ obstacles in the plane. This is the case where all rectangles are considered as obstacles. As a result, we have:

Property2: The construction of the network N=(V, E, d) takes $O(n_1 \times n_2 \times (n_1 + n_2))$ time.

Now, using the properties 1 and 2, theorem 2 gives the complexity of the geometric approach for scheduling the two-job shop problem with blocking.

Theorem 2: The multi-resource two job-shop with blocking problem has a complexity O ($n_1 \times n_2 \times (n_1 + n_2)$), and hence can be solved in polynomial time.

Proof: The theorem follows from properties 1 and 2. As a matter of fact, the determination of resource conflicts and deadlock situations using the dynamic programming takes $O(n_1 \times n_2)$ time. The construction of the network N = (V, E, d) takes O ($n_1 \times n_2 \times (n_1 + n_2)$) time. The determination of the shortest path in an acyclic graph takes $O(|E|+|V|)$ time. It can be shown that, in the worst case, $|V| = n_1 \times n_2$ and $|E| = |V| \times (n_1 + n_2)$. As a result, the overall complexity of our algorithm is O ($n_1 \times n_2 \times (n_1 + n_2)$). Q.E.D.

Theorem 2 gives the complexity in the worst case. Possible refinements of the construction of the network are conceivable. For example, since the upper and left boundaries of an obstacle are forbidden, all adjacent obstacles are considered as a single one, and hence the number of nodes (arcs) of the network can be reduced.

5. A greedy algorithm for scheduling general MJSB

5.1 A combined job approach

In this section, a heuristic algorithm for solving the general MJSB with N jobs is presented. The heuristic algorithm is a greedy method based on the geometric approach for the two-job problem. It considers jobs one by one. At each step, a new job J_i is considered and the algorithm tries to find the best schedule by taking into account schedule of the jobs already considered. After finding the schedule of a job J_i, a combined job J_{com} that represents the schedule of all jobs already considered is defined. At the next step, the combined job and another new job to be examined provide an instance of the two-job problem that can be solved by a modified geometric approach to be proposed in Section 5.2. After examining all jobs in the set $\Im$, the input sequence of operations into the resources and the makespan can be determined from the final combined job J_{com}. The heuristic algorithm can be summarized as follows:

Algorithm 1:

1. Choose a sequence of jobs. Without loss of generality, let the sequence be $J_1, J_2, \ldots, J_n$.
2. Set $J_{com} \leftarrow J_1$.
3. For i = 2 to N do
   3.1. Solve the 2-job problem with J_{com} and J_i.
   3.2. Construct a combined job J' of J_{com} and J_i.
   3.3. Set $J_{com} \leftarrow J'$.
4. The final schedule and C_{Max} can be determined from the combined job J_{com}.

End.

Having solved the two-job problem in step (3.1), the combined job is constructed as follows: Let C_{Max} be the makespan of the two-job problem. We decompose the time interval $[0, C_{Max}]$ into sub-intervals $[t_0, t_1], [t_1, t_2], \ldots, [t_{u-1}, t_u]$ such that no operation starts or terminates inside an interval, i.e. the starting times of operations $S_{ik} \in \{t_0, t_1, \ldots, t_u\}$ for all job J_i and operation O_{ik}. The new combined job J' is a job of *u* operations. Each operation $O_{J',k}$ of job J', for $1 \leq k$

$\leq u$ requires $t_k - t_{k-1}$ time units and all resources of operations of J_{com} and J_i that are being performed in interval $[t_{k-1}, t_k]$. For reasons that will become clear, a total order is determined for all elementary operations of a combined operation.

To illustrate this construction of a combined job, let us consider the following example. The manufacturing processes of jobs are $J_{com} = J_1 = \{(R_1, 1), (R_2, 1)\}$ and $J_i = J_2 = \{(R_2, 1), (R_3, 2)\}$. The geometric approach gives the feasible schedule $\{S_{11} = 0, S_{12} = 1, S_{21} = 0, S_{22} = 1\}$ with the makespan $C_{Max} = 3$. According to the starting and completion times of operations, the makespan can be decomposed into 3 sub-intervals $[0,1]$,$[1,2]$ and $[2,3]$ (see the Gantt diagram of Figure 9). The combined job is $J' = \{(R_1, R_2, 1), (R_2, R_3, 1), (R_3, 1)\}$.

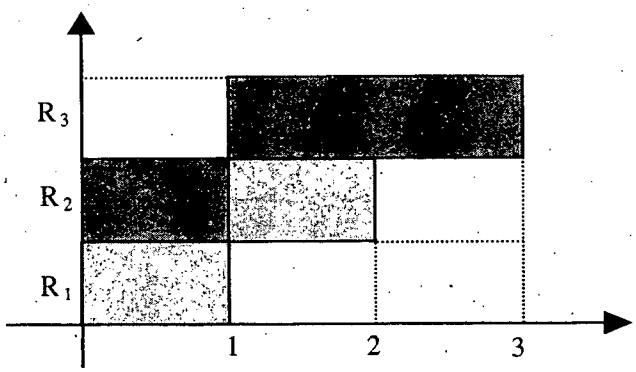


Figure 9. Gantt diagram

5.2 Modified geometric approach

The geometric approach for solving the two-job MJSB will be modified to solve the two-job problem between a combined job J_{com} and a new job J_i. This modification is a direct consequence of the notion of combined operations. As a result, the notion of obstacles needs to be extended.

A. Differences of combined jobs vs. simple jobs

Before presenting the modified geometric approach, let us show by examples problems that may arise when scheduling a combined job J_{com}.

Example 1: Consider an MJSB of three jobs J_1, J_2 and J_3 with the following manufacturing processes:

$$J_1 = \{(R_3, 2), (R_1, 1), (R_2, 1)\},$$
$$J_2 = \{(R_3, 2), (R_2, R_3, 1), (R_1, 1)\},$$
$$J_3 = \{(R_4, 1), (R_2, 3), (R_3, 1)\}.$$

Resources R_1 and R_4 are available in a single unit while there are two units of R_2 and two units of R_3. The jobs are considered in order J_1, J_2, J_3.

Applying the geometric approach to jobs J_1 and J_2 leads to the combined job $J_{com} = \{(O_{11}O_{21}, 2), (O_{12}O_{22}, 1), (O_{13}O_{23}, 1)\} = \{(2R_3, 2), (R_1R_2R_3, 1), (R_1R_2, 1)\}$. Notice that, to pass from the second operation (O_{12} O_{22}) of J_{com} to its third operation (O_{13} O_{23}), job J_1 must progress to operation O_{13} first and then J_2 can progress to operation O_{23}. The reverse is not possible.

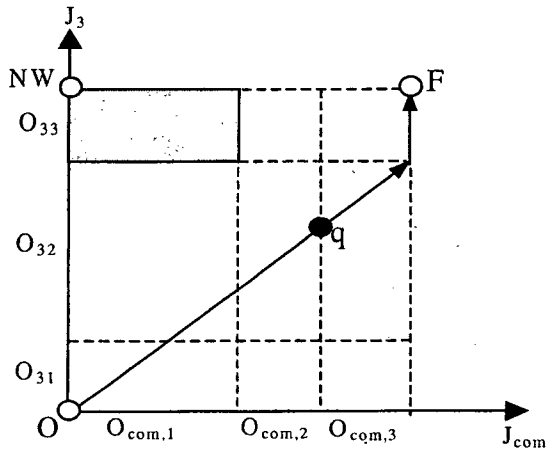


Figure 10. Problem arisen with the combined job

To schedule J_{com} and J_3, the obstacles and the shortest path from O to F are given in Figure 10. Note that at point q, the system is still in state $(O_{com,2}, O_{32})$, i.e. (O_{12}, O_{22}, O_{32}). To proceed, J_{com} should proceed to its third operation while J_3 remains at its second operation O_{32}. The single way to progress to state $(O_{com,3}, O_{32})$ is to let job J_1 pass to operation O_{13}. This is not possible, since resource R_2 needed to start O_{13} is now held by job J_3 for O_{32}. ■

Remark 1: Border crossing of type $(O_{com,k}, O_{ik'})$ to $(O_{com,k+1}, O_{ik'})$ that is possible when J_{com} is considered as a simple job may not be feasible. This is because resources held by the additional job J_i for $O_{ik'}$ may be needed for jobs composing J_{com} to progress.

Remark 2: The above phenomenon does not occur in systems where all resources are available in a single unit. In fact, if it is not possible to progress to state $(O_{com,k}, O_{ik'})$ then the operation $O_{com,k}$ share common resources with operation $O_{ik'}$. Consequently, the sate $(O_{com,k}, O_{ik'})$ is avoided by the dynamic programming approach.

Example 2: Consider another MJSB of three jobs defined as follows:

$$J_1 = \{(R_1, 1), (2R_2, 1)\},$$
$$J_2 = \{(R_2, 1), (R_3, 1)\},$$
$$J_3 = \{(R_3, 1), (R_1, 1)\}.$$

where resources are available in quantity $Q_1=1$, $Q_2=3$, $Q_3=1$. Jobs are considered in order J_1, J_2, J_3. The geometric approach for J_1 and J_2 leads to the combined job $J_{com} = \{(R_1, R_2, 1), (2R_2, R_3, 1)\}$. At the second step, J_{com} and J_3 are considered. The obstacles are given in Figure 11. From this representation, J_3 and J_{com} should be performed sequentially and the schedule takes 4 time units.

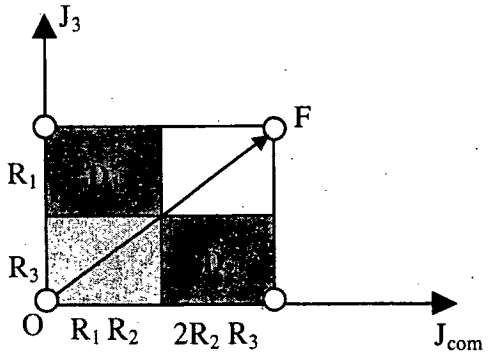


Figure 11. Geometric approach representation of example 2

A detailed examination of the combined job shows that it is possible to pass directly from state ($O_{com,1}$, O_{31}) to state ($O_{com,2}$, O_{32}). To realize this crossing, it suffices to let J_1 progress to O_{12} first, then J_3 progress to O_{32} and finally J_2 progress to O_{22}. As a result, state ($O_{com,1}$, O_{31}) is not a deadlock state and the shortest path which takes 2 time units is given in Figure 11. ■

Remark 3: Contrary to the two-simple-job case where diagonal crossing from state ($O_{1,k1}$, $O_{2,k2}$) to ($O_{1,k1+1}$, $O_{2,k2+1}$) passes always by ($O_{1,k1+1}$, $O_{2,k2}$) or ($O_{1,k1}$, $O_{2,k2+1}$), the diagonal crossing ($O_{com,k1}$, O_{ik2}) to ($O_{com,k1+1}$, O_{ik2+1}) needs not pass by ($O_{com,k1+1}$, O_{ik2}) or ($O_{com,k1}$, O_{ik2+1}) and it can be realized directly by first letting some jobs of J_{com} progress, then letting J_i progress and finally letting the remaining jobs of J_{com} progress. As a result, some deadlock states when J_{com} is considered as a simple job are actually admissible.

B. Modified deadlock characterization

Based on the above remarks, we introduce the following indicators of border crossing:

$$P_x(k_1,k_2)=\begin{cases}1, & \text{if the passage from state } (O_{com,k_1-1}, O_{i,k_2}) \text{ to} \\ & \text{state } (O_{com,k_1}, O_{i,k_2}) \text{ is not possible,} \\ 0, & \text{otherwise.}\end{cases}$$

$$P_y(k_1,k_2)=\begin{cases}1, & \text{if the passage from state } (O_{com,k_1}, O_{i,k_2-1}) \text{ to} \\ & \text{state } (O_{com,k_1}, O_{i,k_2}) \text{ is not possible,} \\ 0, & \text{otherwise.}\end{cases}$$

$$P_{xy}(k_1,k_2)=\begin{cases}1, & \text{if the passage from state } (O_{com,k_1-1}, O_{i,k_2-1}) \text{ to} \\ & \text{state } (O_{com,k_1}, O_{i,k_2}) \text{ is not possible,} \\ 0, & \text{otherwise.}\end{cases}$$

The introduction of $P_{xy}(k_1,k_2)$ is motivated by remark 3. The computation of the above indicators is a deadlock detection problem which is NP-hard in general. To overcome this complexity, we introduce the following assumption:

> *Assumption of combined job:* To each combined operation $O_{com,k}$ is associated a sequence which defines the order at which operations composing $O_{com,k}$ start.

In the greedy algorithm, the above sequences are updated progressively at each step. After scheduling a new job J_i, the new combined job is computed. To each new combined operation $O_{com,k}$, the sequence is obtained by inserting related operation of J_i into the actual sequence of operation $O_{com,k}$.

Remark 4: the computation of $P_x(k_1, k_2)$ is determined simply by verifying deadlocks by letting jobs of J_{com} progress according to the sequence associated to $O_{com,k1}$. $P_y(k_1, k_2)$ is determined by simply checking the resource capacity constraint of state $(O_{com,k1}, O_{i,k2})$. Finally, the computation of $P_{xy}(k_1, k_2)$ is similar to that of $P_x(k_1, k_2)$ by inserting operation $O_{i,k2}$ into the sequence associated to $O_{com,k1}$.

Similar to the MJSB geometric approach, we consider the following indicators for resource conflicts and deadlocks:

$$I(k_1,k_2)=\begin{cases}1, & \text{if operations } O_{com,k_1} \text{ and } O_{ik_2} \text{ have resource conflict,}\\ & \text{or a deadlock situation will be reached starting}\\ & \text{from state } (k_1,k_2),\\ 0, & \text{otherwise.}\end{cases}$$

Using indicators P_x, P_y, P_{xy},

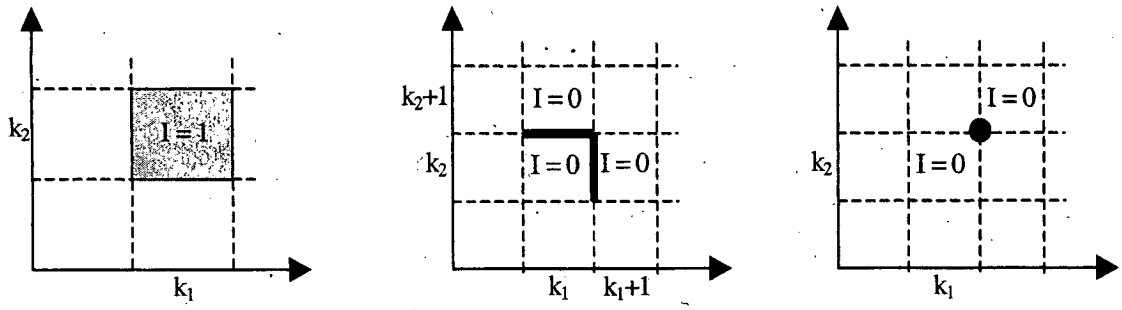
$$I(k_1,k_2)=\begin{cases}1, & \text{if } B^r_{com,k_1}+B^r_{i,k_2}>Q_r \text{ for some r,}\\ & \left(I(k_1,k_2+1)\vee P_y(k_1,k_2+1)\right)\\ & \wedge\left(I(k_1+1,k_2)\vee P_x(k_1+1,k_2)\right)\\ & \wedge\left(I(k_1+1,k_2+1)\vee P_{xy}(k_1+1,k_2+1)\right), \quad \text{otherwise}\end{cases}$$

with $I(k_1, n_2+1) = I(n_1+1,k_2) = 0, \forall k_1, k_2$.

C. Modified obstacle construction

Three types of obstacles need to be considered:

- 2-dimension obstacles which are rectangles corresponding to intervals $O_{com,k1}$ and $O_{i,k2}$ such that $I(k_1, k_2)=1$. These 2-D obstacles correspond to resource conflicts or deadlock situations,
- 1-dimension obstacles which are upper boundaries or right boundaries of a rectangle (k_1, k_2) such that $I(k_1, k_2) = I(k_1, k_2+1) = 0$ and $P_y(k_1, k_2+1) = 1$ or $I(k_1, k_2) = I(k_1+1, k_2) = 0$ and $P_x(k_1+1, k_2) = 1$. These 1-D obstacles are results of remark 1. These obstacles cannot exist if J_{com} is considered as a simple job.
- 0-dimension obstacles which are NE corners of rectangles (k_1, k_2) such that $I(k_1, k_2) = I(k_1+1, k_2+1) = 0$ and $P_{xy}(k_1+1, k_2+1) = 1$. Such obstacles forbid the diagonal crossing into rectangle (k_1+1, k_2+1).



(a) 2-D obstacle (b) 1-D obstacle (c) 0-D obstacle

Figure 12. Three types of obstacles

Figures 13 and 14 show the obstacles of examples 1 and 2. For example 1, an 1-D obstacle is introduced since it is not possible to progress from state ($O_{com,2}$, O_{32}) to ($O_{com,3}$, O_{32}). For example 2, rectangle ($O_{com,1}$, O_{31}) is not a deadlock state since diagonal crossing ($O_{com,1}$, O_{31}) to ($O_{com,2}$, O_{32}) is possible.

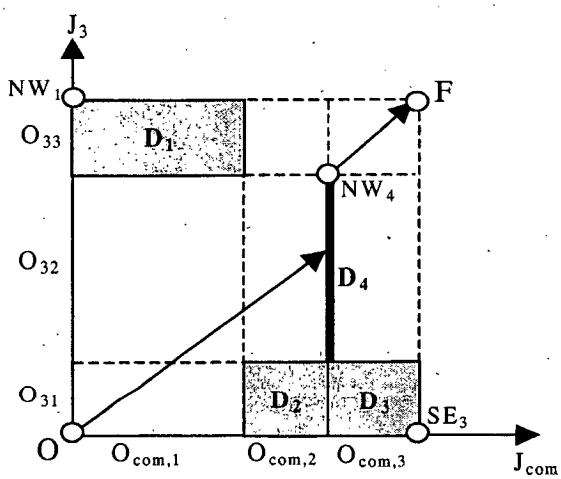


Figure 13. Modified geometric representation of example 1

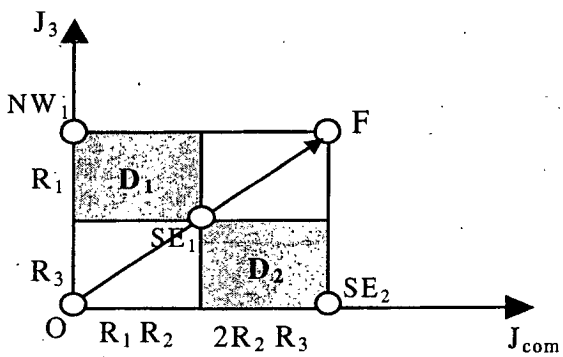


Figure 14. Modified geometric representation of example 2

The following should be taken into account for the construction of the network:

- The upper boundary, the right boundary and the interior of a 2-D obstacle are forbidden,
- An 1-D obstacle is a degeneration case of a 2-D obstacle,

- NW and SE corners of 1-D and 2-D obstacles are nodes of the network,
- A 0-D obstacle always appears at the junction point of two high dimension obstacles (see Figure 15). As a result, if a NW/SE corners of 1-D and 2-D obstacles is a 0-D obstacle, then the corresponding NW/SE corner is forbidden and cannot be a node of the network.

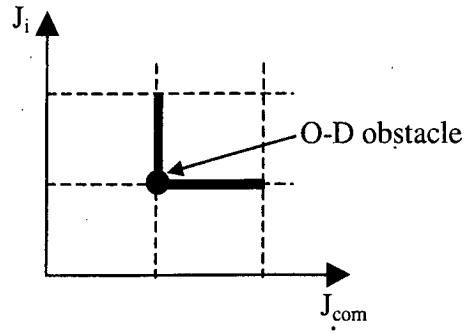
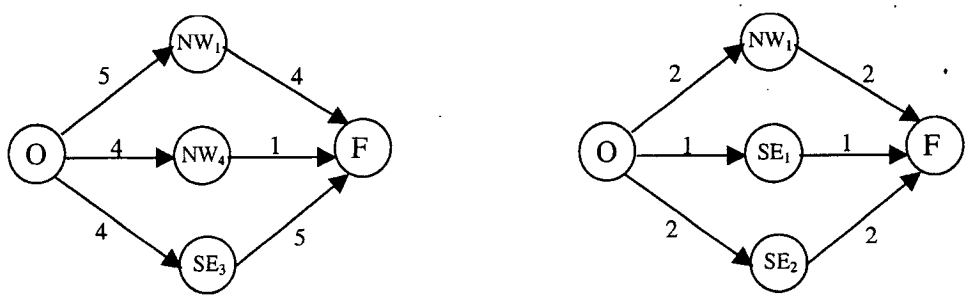


Figure 15. O-D obstacles

For the two examples, the resulting networks are given in Figure 16.



(a) Network of example 1

(b) Network of example 2

Figure 16. Networks of examples 1 and 2

5.3 Improving a given schedule

Although the notion of combined job J_{com} allows to reduce the complexity of the scheduling problem, it always leads to sub-optimality. The main reason is that all operations $\{O_1, O_2, \ldots, O_{i'k'}, O_p\}$ of a combined operation $O_{com,k}$ are treated as a single operation. Due to this fact, an operation $O_{i,k2}$ of the new job J_i, having resource conflicts with some operation $O_{i',k'}$ of the combined operation $O_{com,k1}$, will delay all operations of the combined operation if $O_{i,k2}$ is scheduled before $O_{i',k'}$. In other words, the schedule given by the algorithm 1 may be a delay schedule and some operations are purposely delayed.

To obtain a non-delay schedule, we first extract the input sequence σ_r of all operations into each resource r. This sequence is unique and can be easily determined as a result of the "combined job assumption" that associates to each combined operation a sequence.

We then schedules the operations O_{ik} for all job J_j one after another according to the input sequences σ_r. Note that, when an operation O_{jk} is considered, all operations sharing resource with O_{jk} and preceding O_{jk} in the input sequences σ_r have been scheduled. It is then possible to determine the release epochs of all resources r. Let *Release(r, n)* be the time at which n

units of resource r become available. Finally, the earliest starting time of operation O_{jk} with respect to input sequences σ_r is as follows:

$$S^N_{jk} = \text{Max}\left\{S^N_{j,k-1} + p_{j,k-1}, \text{Max}_{r\in R_{jk}}\left\{Release\left(r, B^r_{jk}\right)\right\}\right\} \quad (1)$$

The greedy algorithm that combines both the combined job approach and the improvement described here is as follows:

Algorithm 1' (Greedy algorithm with respect to a job sequence π)

1. Set $J_{com} \leftarrow J_{\pi(1)}$.
2. For i = 2 to N do
   2.1. Solve the two-job problem with J_{com} and $J_{\pi(i)}$ using the modified geometric approach.
   2.2. Construct a combined job J' of J_{com} and $J_{\pi(i)}$ according to the optimal schedule.
   2.3. Set $J_{com} \leftarrow J'$.
3. Extract the input sequence σ_r of operations into each resource r from the final combined job J_{com}.
4. Compute the earliest starting times S^N_{jk} of all operations using equation (1).

End.

6. A taboo search heuristic

Of course, the quality of the schedule obtained by the greedy algorithm proposed in Section 5 depends strongly on the job sequence chosen in step 1. If the number of jobs is smaller than 7, one can allow to check all possible combinations (7! = 5040). Beyond this limit, the number of possible combinations grows exponentially, and the need of an approach that explores a larger number of interesting job sequences without repetition is vital. The objective is to find the job sequence that leads to the schedule with the shortest makespan.

In this paper a taboo search heuristic is used to explore the solution space of the job sequence. However, as the evaluation of a large number of job sequences using the greedy algorithm is rather time consuming, we introduce a pertinent "artificial objective function" to guide the taboo search.

To achieve this objective, we construct an "artificial" optimization problem, which is equivalent to a single machine-scheduling problem with sequence-dependent setup times. The problem may be described as follows: A collection of N tasks, each arriving at time 0 is to be scheduled on an available machine. Each task i (i = 1, 2,..., N) corresponds to a job J_i of our model, and requires 0 units of time on the machine; S_{ij} is the setup time of scheduling task *j* immediately after task *i*. This setup time is calculated as follows:

$$S_{ij} = \sum_{k=1}^{n_i} p_{ik} + \sum_{k=1}^{n_j} p_{jk} - C_{Max_{ij}}$$

where: C_{Max}_{ij} is the makespan obtained by the geometric approach when solving the two-job problem between J_i and J_j. The quantity S_{ij} can be interpreted as the profit of sequencing job J_i just before job J_j. For each job sequence $\prod = \{\pi(1), \pi(2), ..., \pi(N)\}$ where: $\pi(i)$ is the task in position *i*, the artificial objective function to be maximized is the total profit given by:

$$\text{Gain} = \sum_{i=1}^{N-1} S_{\pi(i)\pi(i+1)}$$

After initializing the setup time's matrix S, the evaluation of the "artificial objective function" of job sequences is very simple.

The taboo search heuristic is introduced by Glover (1986); it constitutes the only local search heuristic that has been explicitly developed with a set of memories. These memories give the method the ability to overcome the cycling phenomenon, and the intelligent exploration of the solution space without being trapped in local optima. Like any local search heuristic, TS starts with the systems in a known solution (selected randomly or by any other heuristic), and try to improve it iteratively until the stopping criterion is satisfied. The iterative improvement consists of a sequence of local perturbations (moves) of the actual solution. The solution obtained after the perturbation becomes the new solution of the system, and the process is continued until no improvements cannot be found. The last situation represents usually a local but not a global optimum. In order to avoid being trapped by this local optimum, TS accept to move into non-improving solutions. Consequently, this concept increase the possibility to return to solutions recently visited, and may, of course, introduce cycling in the algorithm. To alleviate this inconvenience, a taboo restriction that classifies some perturbation as prohibit is employed. These perturbations are recorded in a taboo list, and are classified as taboo for the next T iterations. T is called the size of the taboo list. Finally, the taboo status of the perturbations recorded in taboo list is not absolute, and can be dropped if some conditions are satisfied as expressed by the aspiration criteria.

The components cited above constitute the simple form of the taboo search and when adequately chosen can provide competitive results for combinatorial optimization problems. For the comprehensive examination of this approach, and its associated strategies for searching solutions space by creating and exploiting adaptive forms of memories, the reader is referred to (Glover et al. 1993, Glover, 1995). Now, we give the parameters of TS used in our implementation.

The initial solution is constructed through an insertion heuristic. More precisely, let E_1, E_2 be two subsets that represent respectively jobs examined and not yet examined. Initially, subset E_1 contains the two jobs that give the minimum makespan over all possible pair of jobs, and the combined job J_{com} is then constructed. At each step, a job $J_i \in E_2$ that gives the minimum makespan when combined with J_{com} is selected and inserted in E_1. The combined job J_{com} is updated. The initialization step ends when the set E_2 is empty, and the order in which jobs are inserted in E_1 represents the initial job sequence.

The swap move, which exchanges a job J_i in position $\pi(i)$ and another job J_j in position $\pi(j)$ is adopted. Jobs J_i and J_j are the attributes of this move, and are used to create the taboo restriction. More precisely, after executing the swap, any move that returns job J_i (resp. J_j) in

position $\pi(i)$ (resp. $\pi(j)$) is classified as taboo. This restriction is reinforced by recording these informations in the taboo list.

The dynamic aspect of the size of taboo list is adopted and the size is randomly selected in the reference interval [N/2,N], where N is the number of jobs. The size of the list is updated every 2N iterations to give the approach the ability to work with other sizes. The well known global aspiration criteria that overrides the taboo status of a move whenever moving it result in a new best solution is used. Finally, the algorithm is stopped after a maximum number of iterations fixed a priori is reached. In this application the maximum number of iterations is fixed at 5000.

In the following we summarizes the methodology used, and gives the pseudo code for the final *TSGA algorithm* proposed for the MJSB.

Algorithm 2: *TSGA algorithm*

1. *Initialization*: Determine the initial job sequence by using the insertion heuristic. Let the sequence be $J_{\pi(1)}, J_{\pi(2)},\ldots, J_{\pi(N)}$. Set the best makespan $C^*_{Max} = \infty$.
2. Compute a schedule J_{com} using the greedy algorithm 1' with job sequence π. Let C_{max} be its makespan.
3. If $C_{max} < C_{max}^*$, then set $C_{max}^* \leftarrow C_{max}$ and record the schedule J_{com} as the best schedule.
4. *Stopping criterion*: If the maximum number of iterations is reached, then stop.
5. Find the best not taboo swap move according to the "artificial objective function".
6. Execute the move to create a new job sequence π.
7. Update the taboo structures, i.e. taboo list and taboo size.
8. Go to 2.

End.

7. Computational results

Unlike the classical job shop, the benchmark instances for the multi-resource job shop problem with blocking are limited. In this section, we present numerical results for 12 existing instances.

The three first instances are from a real automated manufacturing workstations (Ramaswamy and Joshi, 1996) with three machines and a robot to handle the transportation of parts from one machine to another, loading and unloading the parts. There are four jobs to be scheduled. In problem RJ1, parts can move automatically from one machine to the next one and the robot is not used. In problem RJ2, the robot is needed for moving a part from one place to another place. In problem RJ3, there is one buffer space between any two machines and the robot is not used to handle parts. For the three cases, optimal solutions are known. A fourth instance MDX1 of multiple resources is considered as well. MDX1 is similar to instance RJ2 except that two transportation robots instead of one are used.

The remaining instances are from (Damasceno, 1999), which include five instances with unitary resources and three instances with multiple resources. The instances with unitary resources contain two instances with nine jobs and three instances with ten jobs, while the three remaining instances with multiple resources contain all ten jobs. These instances were randomly generated as follows: The number of operations of each job is selected randomly in

interval [3, 5]. The number of resources for each operation is selected from interval [1, 3], and type of resources needed is chosen randomly from the available resources. The processing time of each operation is selected randomly in the interval [1, 10]. Further, for instances with multiple resources, the available quantity of each resource is selected from interval [1, 5]. Instances considered in this section are summarized in Table 1. The difference of the operation numbers of the three first instances is due to the consideration of transportation and waiting in buffer.

Table 1. Numerical examples

Problem	Number of jobs	Number of operations per job	Number of resources	Units of each resource	Number of resources for each operation	Processing time
RJ1	4	3	3	1	1	[32, 212]
RJ2	4	7	4	1	1	[3, 212]
RJ3	4	5	6	1	1	[0, 212]
UDX1	10	[3, 5]	10	1	[1, 3]	[1, 10]
UDX2	10	[3, 5]	9	1	[1, 3]	[1, 10]
UDX3	9	[3, 5]	9	1	[1, 3]	[1, 10]
UDX4	10	[3, 5]	8	1	[1, 3]	[1, 10]
UDX5	9	[3, 5]	8	1	[1, 3]	[1, 10]
MDX1	4	7	4	[1, 2]	1	[3, 212]
MDX2	10	[1,3]	5	[2, 4]	[1, 3]	[1, 10]
MDX3	10	[1,3]	6	[1, 3]	[1, 3]	[1, 10]
MDX4	10	[1, 5]	10	[1, 3]	[1, 3]	[1, 10]

Our solutions are compared with the solutions proposed by (Damasceno, 1999). This last heuristic is one of the few approaches that treat the same characteristics of our model. The results of five independent runs are given in Table 2.

Table 2. Results of 5 independent runs on existing instances

Problem	TSGA algorithm			Damasceno, 1999	
	Worst	Best	CPU secs	Makespan	CPU secs
RJ1	512	**512***	0.02	n.g	n.g
RJ2	560	**560***	0.02	568	31
RJ3	502	**502***	0.02	n.g	n.g
UDX1	90	**90**	1.39	98	n.g
UDX2	109	**109**	4.15	124	n.g
UDX3	103	**103**	8.41	115	n.g
UDX4	112	**112**	7.53	126	n.g
UDX5	104	**103**	0.09	122	n.g
MDX1	451	**451**	0.02	**451**	31
MDX2	33	**33**	0.37	43	n.g
MDX3	35	**35**	0.03	44	n.g
MDX4	47	**47**	5.16	56	n.g

*: Optimal solution
n.g: Not given

For the three first instances, the *TSGA algorithm* is able to obtain the optimal solutions. For the other problems, our approach always outperforms the approach of (Damasceno, 1999),

and allows to give new best makespan for all instances. Furthermore, these solutions are obtained in relatively short computation time. Finally, by observing the results given in Table 2, it appears that the *TSGA algorithm* is very stable for independent runs.

8. Conclusion

Scheduling and deadlock avoidance problems of a class of automated manufacturing systems are considered in this paper. A new scheduling model is presented and special cases are discussed. The model is characterized by complex product structures and complex resources behaviors, which are very often encountered in practice. An algorithm based on a new extension of the well-know geometric approach for the two-job shop problems, and a taboo search heuristic is developed. This algorithm shows that it is possible to elaborate efficient approaches for deadlock-free scheduling problems without separating the deadlock detection and the scheduling problem. Numerical experiment shows that the proposed algorithm is efficient – it has obtained optimal schedules for some benchmarks and new best makespan for other existing benchmarks –, and fast – the schedule are obtained in short execution time.

Future research will be conducted in two directions. The first direction is the optimization of other criteria. The second direction is the development of intelligent approaches to solve the more general systems including routing flexibility and assembly/disassembly operations.

References

Banaszak, Z.A., B.H. Krogh (1990). Deadlock Avoidance in Flexible Manufacturing Systems with Concurrently Competing Process Flows. *IEEE Trans. Robot. Automat.*, **6**, 724-734.

Blazewicz, J. and G. Finke (1994). Scheduling with Resource Management in Manufacturing Systems. *European J. Operational Research*, **76**, 1-14.

Blazewicz, J., W. Domschke, and E. Pesch (1996). The Job Shop Scheduling Problem: Concentional and New Solution Techniques. *European J. Operational Research*, **93**, 1-33.

Brucker, P. (1988). An Efficient Algorithm for the Job-Shop Problem with Two Jobs. *Computing*, **40**, 353-359.

Brucker, P. (1998). *Scheduling Algorithms*. Springer-Verlag, Berlin Heidelberg.

Camus, H. (1997). Conduite de systèmes flexibles de production manufacturiere par composition de regimes permanents cyclics : modélisation et évaluation de performances à l'aide des réseaux de Petri. *PhD Thesis*, Ecole centrale de Lille, France.

Chu, F. and X.L. Xie (1997). Deadlock analysis of Petri nets using siphons and mathematical programming. *IEEE Trans. on Robotics and Automation*, **13**, 793-804.

Damasceno, B.C. and X.L. Xie (1998). Scheduling and deadlock avoidance of a flexible manufacturing system. *Proc. IEEE Conf. on Systems, Man and Cybernetic*, pp. 564-569, San Diego, USA.

Damasceno, B.C. and X.L. Xie (1999). Petri Nets and Deadlock-free Scheduling of Multiple-Resource Operations. *Proc. IEEE Conf. on Systems, Man and Cybernetic*, Tokyo, Japan.

Damasceno, B.C. (1999). Ordonnancement des Systemes de Production Multi-ressources avec la Prise en Compte de Blocage, *PhD Thesis*, Université de Metz, France.

Dauzère-Pérès, S. and J. Paulli (1997). An Integrated Approach for Modeling and Solving the Multiprocessor Job-Shop Scheduling Problem Using Tabu Search. *Annals of Operations Research*, **70, 281-306.**

Glover, F. (1986). Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers and Operations Research*, **13**, 533-549.

Glover, F., E. Taillard and D. de Werra. (1993). A User's Guide to Tabu Search. *Annals of Operations Research*, **41**, 3-28.

Glover, F. (1995). Tabu Search Fundamentals and Uses. *Revised and Expanded, Technical Report*, Graduate School of Business, University of Colorado, Bolder, CO.

Ezpeleta, J., J.M. Colom, and J. Martinez (1995). A Petri Net Based Deadlock Prevention Policy for Flexible Manufacturing Systems. *IEEE Trans. Robot. Automat.*, **11**, 173-184.

Fanti, M.P., B. Maione, S. Mascolo, and B. Turchiano (1997). Event-Based Feedback Control for Deadlock Avoidance in Flexible Production Systems. *IEEE Trans. Robot. Automat.*, **13**, 347-363.

GOTHA (1993). Les Problèmes d'Ordonnancement. *Recherche Opérationnelle/Operations Research*, **27**, 77-150.

Hall, N.G. and C. Sriskandarajah (1996). A Survey of Machine Scheduling Problems with Blocking and No-Wait In Process. *Operations Research*, **44**, 510-525.

Hillion, H., J.M. Proth and X.L. Xie (1987). A Heuristic Algorithm for the Periodic Scheduling and Sequencing Job-Shop Problem. *Proc. IEEE Conf. on Decision and Control*, Los-Angeles.

Jeng, M.D., S.C. Chen and C.S. Lin (1996). A Search Approach Based on the Petri Nets Theory for FMS Scheduling. *Proc. 13th IFAC World Congress*, San Francisco, vol. B, 55-60.

Lee, D.Y. and F. DiCesare (1994). Scheduling Flexible Manufacturing Systems Using Petri Nets and Heuristic Search," *IEEE Trans. Robot. Automat.*, **10**, 123-132.

Proth, J.M. and X.L. Xie (1996). Petri nets: A tool for design and management of manufacturing systems. John Wiley & Sons.

Ramaswamy, S.E. and S.B. Joshi (1996). Deadlock-free schedules for automated manufacturing workstations. *IEEE Trans. on Robotics and Automation*, **12**, 391-400.

Sethi, S.P., C. Sriskandarajah, G. Sorger, J. Blazewicz, and W. Kubiak (1992). Sequencing of Parts and Robot Moves in a Robotic Cell. *Int. J. Flexible Manuf. Sys.*, **4**, 331-358.

Unité de recherche INRIA Lorraine
Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - B.P. 101 - 54602 Villers lès Nancy Cedex (France)

Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot St Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - B.P. 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, B.P. 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399

ISRN INRIA/RR--4137--FR+ENG